Intersection Optimization is NP Complete

Guillaume Bonfante[1] and Joseph Le Roux[2]

[1] INRIA - LORIA
[2] INPL - Nancy 2 - LORIA

Abstract. We propose a method for lexical disambiguation based on polarities found in several grammatical formalisms which require multiples intersection of finite state automata. We then prove that there is no efficient technique to minimize the state complexity of these intersections.

1 Introduction

The main concern of this paper is to answer the following question: given a set $\{A_1, \ldots, A_k\}$ of finite state automata, can we guess an order on them to perform efficiently their intersection? More precisely, can we find a permutation π for which the following algorithm will run as fast as possible ?

```
A = A[pi[1]];
for i = 2 to k do
   A = A intersect A[pi[i]]
done
```

Observe that computing the intersection as above takes in the worst case exponential time. Indeed, the size of the result $|\bigcap_{i\leq k} A_i| = \prod_{i\leq k} |A_i|$ is exponential. We refer to Saaloma and Yu to learn more about state complexity [1, 2]. But this is not the issue addressed here. The question is to find the order in which we have to perform the intersections. And we show that this part of the problem is also inherently difficult. To get rid of the size problem, we consider the ordering problem with regards to some a priori upper bound on the size of automata. It results that the decision problem is NP-complete.

A standard NP-complete problem about intersection of automata is the emptiness of the result. See for instance [3], or [4] who gives explicit upper bounds. But, here, we are more concerned by the intersection process than by the result itself. A question analogous to our present issue is about matrix multiplication. Given a sequence of matrices $M_1, \ldots, M_k$ of different size. The way one parenthesizes the expression $M_1 \times \cdots \times M_k$ has a huge impact on the cost of evaluating the product (see [5]). For this problem, computing the good order can be done in polynomial time by a dynamic programming procedure.

Let us now present the practical application for which the question aroused as a motivation for the present study. It deals with disambiguation of lexicalized polarized grammars (PGs) like Categorial Grammars [6] (CGs), Interaction

Grammars [7] (IGs) or Polarized Unification Grammars of Kahane [8] (PUGs). A lexicalized grammar is defined by its lexicon, which associates a set of *syntactic items* to every word of the language; each of these items specifies a grammatical construction in which the corresponding word participates. There is a complexity issue if one considers exact parsing with large scale lexicalized grammars. Indeed, the number of way of associating to each word of a sentence a corresponding item is the product of the number of lexical entries for each word. The idea of using automata to represent this exponential is not new. The main reason is that the representation of all lexical choices can be done in linear space. In particular, methods based on Hidden Markov Models (HMMs) use such a technique for part-of-speech tagging [9–11]. With such a representation, one may conceive dynamic programming procedures, and consequently, benefits from an exponential temporal speed up as well as the space one.

One of the main features of PGs is that each syntactical item is equipped with positive features, negative features and neutral ones. A positive feature is a resource brought by a word, a negative one is expected by the word. For instance, a determiner provides a noun phrase and expects a common noun. A neutral feature represents a linguistic property which does not behave as a consumable resource. Polarities are used to guide the process of syntactic composition: features with the same type but with opposite polarities try to neutralize themselves mutually. The process ends successfully in the parse structure of a sentence where all polarized features are neutralized.

Syntactic items, if we forget their structure, become bags of polarized features. A necessary condition for a tagging to be successful is that counting polarized features in the bag must end with a zero. Automata are a well-suited way of factorizing this counting of polarities. The problematic point is that one may count different features, each of which provides an automaton. Hence, the resulting necessary condition is given by the intersection of these automata [12].

As it is known [13], when performing multiple intersections, intermediate automata can possibly be huge, even if the final automaton is small. So in the second part of this article we address the issue of the state complexity not of the atomic operation itself, namely intersection, but the state complexity of the whole process in which these atomic operations have to be performed. So, the main issue is to intersect the automata in a right order. To fix the ideas, usually, we have from 10 to 20 automata, each of which has say 20 states. The state complexity theory gives an appalling bound of the order 20^{10}. For our application, we have observed that the size of the resulting automata are somewhat reasonable. But, the choice of the order in which we perform the intersection has a great influence on the speed of the algorithm. We have experienced ratios above 100 between a "good" order and a "bad" order.

We prove that looking for the order in which intersections have to be performed in order to create the minimal number of intermediate states is actually NP-hard. For that reason, we have used heuristics in our implementation.

2 Polarized Grammars and Lexical Disambiguation

In this section, we present a general lexical disambiguation method for PGs relying on automata intersection. For sake of simplicity, we have kept our explanations independent from concrete formalisms.

2.1 Polarized Grammars and Parsing

We give here a very brief description of such grammars. We have adapted it to our current purpose and any reader who wants a wider presentation of the formalism of these grammars should refer to [6–8].

A polarized grammar is equipped with:

- a set W of words (for instance English vocabulary);
- a set S of item (such an entry can be a "noun phrase coordination");
- a function $\ell : W \to P_{\text{fin}}(S)$ which associates any word with a finite set of items;
- a set of feature names $\mathcal{F}$ and a set of feature values $\mathcal{V}$. Examples of feature name are "category" and "gender". Examples of feature value are "noun" or "masculine";
- a function $\rho : S \times \mathcal{F} \times \mathcal{V} \to I[\mathbb{Z}]$ which associates to any item and feature name/value a finite interval over the integers. This function counts the polarized features of items. For instance, $\rho(\text{determiner}, \text{"cat"}, \text{"noun"}) = [-1, -1]$ which says that a determiner waits for exactly one noun. Intervals are summed according to $[a, b] + [c, d] = [a + c, b + d]$.

Given a sentence $w_1, \ldots, w_n$ of words in W, the parsing process consists in a) selecting one item for each word of the sentence, say $s_1, \ldots, s_n$ and b) to check that this selection verifies some properties depending on the grammatical framework. However, there is one common property to all PGs which is: $\forall f, v \in \mathcal{F} \times \mathcal{V} : 0 \in \sum_{i \leq n} \rho(s_i, f, v)$. This property that we call the *global neutrality criterion* reflects the neutrality constraint on final structures.

2.2 Counting with automata

We assume a sentence $w_1 w_2 \ldots w_n$ to parse with a PG G.

Given a feature name and a feature value (f, v), consider the automaton $A(f, v)$ as follows.

- A state of the automaton is a pair (i, p), where i corresponds to the position of the word in the sentence and p is an interval of $\mathbb{Z}$, which represents the counting of polarities.
- Transitions have the form $(i, p) \xrightarrow{s_\alpha} (i + 1, q)$, where $s_\alpha \in \ell(w_i)$, $q = p + \rho(s_\alpha, f, v)$.
- The initial state is $(0, [0, 0])$.
- The accepting states are states (n, p) such that 0 is an element of p.

Every path in $A(v, f)$ from the initial state $(0, [0, 0])$ to an accepting state represents a lexical selection that verifies the global neutrality criterion. Other paths can be deleted. So, any path to an accepting state is a candidate for selection.

Actually, it is a necessary condition for a correct lexical selection to be recognized by the polarity automata, for every *choice* of the feature f and the value v. As a consequence, the intersection of two or more polarity automata gives an automaton which also contains the good solutions. The principle of our selection is to build the automaton[3] $\bigcap_{(f,v)\in\mathcal{F}\times\mathcal{V}} A(f, v)$.

We implemented this disambiguation method in our parsing platform for IGS in the LEOPAR tool[4]. The screenshot given in Appendix come from this software. We illustrate the results for a French[5] sentence *"Il reste la demande."* (the request remains). Figure 1 shows the initial automaton representing all lexical selections provided by an IG for the sentence and figure 2 shows the final automaton after all intersections have been performed whatever the order is. We filtered by intersections with automata from Table 1.

Table 1 present the different filtering automata for disambiguation. Numbers of states, numbers of edges and numbers of paths are given although in the following, we will focus on the number of states. Table 1 shows that while we always end up with the same final automaton, sizes of intermediate automata may vary according to the order in which intersections take place (last two columns). Ordering filtering automata by their size in increasing order seems to be a good heuristic. Although this sentence is very short (4 words and a punctuation mark) we can see that there is a ratio of 1.6 between the two orders presented. On real size sentences with 20 or 30 words, ordering intersections has a dramatic influence on the performance of the computations.

3 NP-Completeness of the problem

In this section, we review three problems, which we prove to be NP-complete, related to our disambiguation technique based on automata intersections.

In these three problems we ask whether it is possible to choose the right order in which the intersection of several automata must be performed in order to minimize the number of intermediate states.

We prove NP-hardness of these problems by reduction from the Traveling Salesman Problem (TSP) [3]. To fix the notations, we first recall this illustrious problem.

An instance of the TSP is a triple (V, d, K) where $V = \{1, \ldots, n\}$ is a set of cities, d is a distance function between any pair of different cities, $d(i, j) \in \mathbb{N}^+$, and a bound $K \in \mathbb{N}^+$. The problem is to decide whether there exists a tour of all

[3] In fact we can restrict our attention to some more restricted values for f and v as shown in Appendix. See [12] for details.

[4] see http://www.loria.fr/

[5] Since our grammar for French is the more complete, it illustrates more correctly the issues of disambiguation. But the question would be the same for English for instance.

cities with a length lesser than K or in other words if there exists a permutation π of the cities such that $(\sum_{i=1}^{i=n-1} d(\pi(i), \pi(i+1))) + d(\pi(n), \pi(1)) \leq K$. To help writing, when π is a function $[1..n] \to [1..n]$ and the context is clear, we write $\pi(n+1)$ for $\pi(1)$ and $\pi(0)$ for $\pi(n)$. So the previous sum can be written $\sum_{i=1}^{i=n} d(\pi(i), \pi(i+1)) \leq K$. From now on, we restrict our attention to those cases where $d(i,j) \leq 2$. The problem remains NP-complete (it corresponds to the reduction from Hamiltonian Circuit).

We will distinguish between the traditional TSP as it has been exposed above and a variant that we call the exact TSP in which the tour must be of length exactly K (see [3]).

3.1 Intersection Optimization Problems

We present a first intersection optimisation problem, that we will enrich to get the second and third problems that are more difficult.

Problem 1. **(IO1)** Let $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ be a set of n finite state automata, $B \in \mathbb{N}^+$ a bound and K a target size. Is there an injective function $\pi : [1..j] \to [1..n]$ such that

- $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| = K$
- for all $k < j$, $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(k)}| \leq B$.

In other words, is there a subset $\mathcal{A} \subseteq \mathcal{A}_n$ such that $|\bigcap_{A \in \mathcal{A}} A| = K$ with all intermediate steps smaller than B?

Problem 2. **(IO2)** Let $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ be a set of n finite state automata and and $B \in \mathbb{N}^+$ a bound. Is there a bijection $\pi : [1..n] \to [1..n]$ such that for any $j \leq n$ we have $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$?

Problem 3. **(IO3)** Let $\mathcal{A}_n = (A_i)_{1 \leq i \leq n}$ be a set of n finite state automata and and $B \in \mathbb{N}^+$ a bound. Is there a permutation π of $[1..n]$ such that $\sum_{1 \leq j \leq n} |(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$?

In the proofs, we do not use automata with loops. So the problems can be stated with or without star languages.

3.2 NP algorithms

These three problems are in NP. Each time we have to choose a permutation π and then:

- for (IO1), if an intermediate intersection is empty we stop and the answer to the problem is "no" (of course, if $K = 0$ it is "yes") if it has size greater than B, the answer is no. Otherwise we proceed to the next intersection. When j intersections have been performed we compare the size of the resulting automaton to K. Observe that those intersections can be performed in time bounded by B^2 since all intermediate steps must have size smaller than B. And so, we are polynomial with regards to B.

– for (IO2), if an intermediate intersection is empty then the answer is "yes" else if it is greater than B (again, we may need to consider B^2 states before minimization) the answer to the problem is "no" else we proceed to the next intersection.
– for (IO3), we need to sum the sizes of the intermediate intersections and check that this sum is never greater than B. If an intersection is empty or if a partial sum exceeds B then we can stop immediately.

3.3 NP-Completeness

Theorem 1. ***(IO1)*** *is NP-complete.*

Proof. We consider some cells, that we will associate to build automata. They are given by Figure 3.

We can note that for $i \in \{0, 1, 2\}$ we have $|C_i| = |C_0| + i$. In other words, these automata encode the "length" of an edge. Observe also that $C_i \cap C_0 = C_i$. So that C_0 is the "neutral" element for the intersection. Finally, if A is some automaton, A' denotes the same automaton, but with primed letters.

Now, given an instance of the exact TSP (V, d, k). We consider a set of automata $\mathcal{A}_{i,j,m}$ with $i, j \in V$ and $m \leq n$ where n is the number of cities in V. We suppose we are given an arbitrary (but minimal) automaton D of size $6 \times n + 3$.

We define

$$\mathcal{A}_{i,j,1} = V_i C_{d(i,j)} V_j (C_0 V_{V\setminus\{i,j\}})^{n-2} C_0 V_i + V'_{V\setminus\{1\}} D'$$

and for $n > m > 1$,

$$\mathcal{A}_{i,j,m} = (V_{V\setminus\{i,j\}} C_0)^{m-1} V_i C_{d(i,j)} V_j (C_0 V_{V\setminus\{i,j\}})^{n-m} + V'_{V\setminus\{m\}} D'$$

and

$$\mathcal{A}_{i,j,n} = V_j C_0 (V_{V\setminus\{i,j\}} C_0)^{n-2} V_i C_{d(i,j)} V_j + V'_{V\setminus\{n\}} D'$$

To fix the intuition, the automaton $\mathcal{A}_{i,j,m}$ correspond to the choice of going from city i to city j at step m. In other words, it corresponds to the choice $\pi(m) = i$ and $\pi(m+1) = j$.

Let us consider the "witness" automaton $\mathcal{A} = (V_V C_0)^n V_V$. Observe that $|\mathcal{A}_{i,j,m}| = |\mathcal{A}| + d(i, j) + |D'|$.

The reduction is then $(V, d, k) \mapsto ((\mathcal{A}_{i,j,m})_{i,j,m}, 2|D'|, |\mathcal{A}| + k)$.

For the correctness, observe that if there is a tour defined by π of length exactly k, then

$$\bigcap_{1\leq m\leq n} \mathcal{A}_{\pi(m),\pi(m+1),m} = V_{\pi(1)} C_{d(\pi(1),\pi(2))} V_{\pi(2)} C_{d(\pi(2),\pi(3))} \cdots C_{d(\pi(n),\pi(1))} V_{\pi(1)}$$

which has size $|\bigcap_{1\leq m\leq n} \mathcal{A}_{\pi(m),\pi(m+1),m}| = |\mathcal{A}| + \sum_{1\leq m\leq n} d(m, m+1)$. The bound on intermediate automata is discussed widely in the following. So, if the TSP problem has a solution, then its encoding has a solution for (IO1).

For the converse part, we consider the set $\mathfrak{A}$ of automata $(\mathcal{A}_{i,j,m})_{i,j,m}$ closed by intersection. Any non empty automata $A \in \mathfrak{A}$ has the following properties (proved by successive inductions):

(i) $A = A_1 + A_2$ with
  - $A_1 = \emptyset$ or
  - $A_1 = V_{\alpha_1} C_{\beta_1} V_{\alpha_2} C_{\beta_2} \cdots V_{\alpha_n} C_{\beta_n} V_{\alpha_{n+1}}$, $\alpha_i \subseteq V$, $\beta_i \in \{0, 1, 2\}$,

  and $A_2 = V_S D'$ with $S \subseteq \{1..n\}$;
(ii) In (i), if $\alpha_j = \{k\}$ for some j, then no other α_ℓ contains k for $\ell \leq n$,
(iii) In (i), $\beta_i = 0$ iff $i \in S$,
(iv) In (i), if $\beta_i \neq 0$, then $\alpha_i = \{k\}, \alpha_{i+1} = \{\ell\}$ are singleton sets and $\beta_i = d(k, \ell)$.
(v) In (i), $\alpha_1 = \alpha_{n+1}$

From (i), we can say that $|A| = |A_1| + |A_2| - 1$ if both part are not empty. Otherwise $|A| = |A_1| + |A_2|$. So, in the worst case, $|A| \leq 2 + \sum_{i=1}^{n} |C_{\beta_i}| + |D'| < 2 \times |D'|$, and the bound on intermediate steps is always respected. From (iii), we learn that $V'_S D'$ is empty iff $\forall j : \beta_j \neq 0$. So that (iv) with (ii,v) gives us the fact (F) that for all i, the set $\alpha_i = \{k_i\}$ is a singleton set and $\pi : [1..n] \to [1..n]$ which sends $i \mapsto k_i$ is a bijection and $k_1 = k_{n+1}$. Since, $|D'| > |\mathcal{A}| + k$, $|A| = |\mathcal{A}| + k$ iff S is empty. The fact (F) above shows that it corresponds to an acceptable tour.

Theorem 2. ***(IO2)*** *is NP-complete.*

Proof. We reduce the TSP to IO2. Let (V, d, k) be an instance of the TSP, Again, let 2 be the maximal distance between two towns and $n = |V|$. Again, for each edge (i, j) with distance $d(i, j)$ we build n automata according to the possible positions of this edge in a tour. That is to say we build n^3 automata $\mathcal{A}_{i,j,p}$. Technically speaking, with regards to what precedes, we must have a stronger control on the order in which the intersection is performed. This is due to the fact that we have a weaker condition that applies to every intermediate automaton. We decompose automata in three components:

1. The first one detailed in Fig. 4 (that we call $\mathcal{C}_{1,i,j,p}$) is responsible for computing the total distance of the tour, like in (IO1) but without indexing V by a set of cities. The end states of the C_0 sub-components are connected to the initial state of $\mathcal{C}_{2,i,j,p}$ by a dummy letter X. Hence, if all the distances are instantiated (as in IO1) then only the last V will connect this first component to the second component.
2. The second one ($\mathcal{C}_{2,i,j,p}$) is responsible for chaining the edges correctly to make a valid tour. This component is shown on Figure 5. It should be observed that if it is intersected with $\mathcal{C}_{2,j,k,p+1}$ then the resulting automaton is of the same size. Otherwise (if city indexes do not match) then it grows by $2n$ states.
3. The third one ($\mathcal{C}_{3,i,j,p}$), presented in Fig. 6, *forbids* (by making any intersection too big) the use of a position more than once and the use of position p without first considering positions $1, \ldots, p-1$. Otherwise it grows by $4n$ states.

Finally we need an additional automaton T, shown on Fig. 7.
The size of $\mathcal{A}_{i,j,p}$ is $|\mathcal{A}_{i,j,p}| = |\mathcal{C}_{1,i,j,p}| + |\mathcal{C}_{2,i,j,p}| + |\mathcal{C}_{3,i,j,p}|$ where

$$|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$$
$$|\mathcal{C}_{2,i,j,p}| = \begin{cases} 6n+2 \text{ if } p = n \\ 4n+2 \text{ otherwise} \end{cases}$$
$$|\mathcal{C}_{3,i,j,p}| = 3(p-1)(4n) + 2(n-p+1)(4n) + 2n = 2n(4n+2p-1)$$

$$|\mathcal{A}_{i,j,p}| = \begin{cases} 2 + d(i,j) + 4n(2+p+2n) & \text{if } 1 \leq p < n \\ 2 + d(i,j) + 2n + 4n(2+3n) \text{ otherwise} \end{cases}$$

We want to prove that $i_1, i_2, \ldots, i_n, i_1$ is a tour for the TSP with length lesser than k if and only if every intermediate automaton of the intersection

$$\bigcap_{1 \leq i \leq m, \alpha(i) \in I \subset [1..n]^3} \mathcal{A}_{\alpha(i)} \cap T \bigcap_{m+1 \leq j \leq n^3, \beta(j) \in [1..n]^3 \setminus I} \mathcal{A}_{\beta(j)}$$

is an automaton whose size is lesser than $B = 2 + k + 4n(1+2n)$.

Preliminary observations Without loss of generality, we can suppose that $k \leq 2n$. Otherwise the TSP is trivial. This entails that, among all the automata $(\mathcal{A}_{i,j,p})_{i,j,p}$, only the automata $(\mathcal{A}_{i,j,1})_{i,j}$ are smaller than B.

i) $|\mathcal{A}_{i,j,1}| = 2 + d(i,j) + 4n(1+2n) < B$
ii) otherwise,

$$\begin{aligned} |\mathcal{A}_{i,j,p}| &\geq 2 + d(i,j) + 4n(1+p+2n) \\ &\geq 2 + d(i,j) + 4n(2+2n) + 4n(p-1) \\ &> 2 + d(i,j) + 4n(1+2n) + k > B \end{aligned}$$

Then, notice that:

iii) $|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$
iv)

$$|\mathcal{C}_{1,i,j,p} \cap \mathcal{C}_{1,k,l,q}| = \begin{cases} 2n + d(i,j) + d(k,l) & \text{if } p \neq q \\ 2n + \max(d(i,j), d(k,l)) & \text{otherwise} \end{cases}$$

v)

$$|\mathcal{C}_{2,i,j,p} \cap \mathcal{C}_{2,k,l,q}| = \begin{cases} |\mathcal{C}_{2,i,j,p}| & \text{if } q = p+1 \text{ and } j = k \\ |\mathcal{C}_{2,i,j,p}| + 2n \text{ otherwise} \end{cases}$$

vi)

$$|\mathcal{C}_{3,i,j,p} \cap \mathcal{C}_{3,k,l,q}| = \begin{cases} |\mathcal{C}_{3,i,j,p}| & \text{if } q = p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n|p-q| & \text{if } q > p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n & \text{if } q \leq p \end{cases}$$

More generally for any sequence prefix of a tour $i_1, i_2, \ldots, i_k$ with $k \leq n+1$ (if $k = n+1$ we force $i_k = i_1$) in our instance of the TSP, we have

$$\begin{aligned} |\bigcap_{1 \leq p \leq k} \mathcal{A}_{i_p, i_{p+1}, p}| &= |\bigcap_{1 \leq p \leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\bigcap_{1 \leq p \leq k} \mathcal{C}_{2,i_p,i_{p+1},p}| + |\bigcap_{1 \leq p \leq k} \mathcal{C}_{3,i_p,i_{p+1},p}| \\ &= |\bigcap_{1 \leq p \leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| \\ &= 2n + (\textstyle\sum_{1 \leq p \leq k} d(i_p, i_{p+1})) + 4n + 2 + 2n(4n-1) \\ &= 2 + (\textstyle\sum_{1 \leq p \leq k} d(i_p, i_{p+1})) + 4n(1+2n) \end{aligned}$$

Correction of the reduction. We first show that if there exists a tour $i_1, i_2, \ldots, i_n, i_1$ with length lesser than k then all the intersections $A_{i_1,i_2,1} \cap \ldots \cap A_{i_j,i_{j+1},j}$ for j ranging from 1 to n are of size lesser than B. We do this by induction on the steps of the intersection process.

As stated earlier, the initial automaton must be $A_{i_1,i_2,1}$ because every other $A_{i_1,i_2,p}$ would be too large. Then, by application of the equality defined in the previous section

$$\begin{aligned} |A = \textstyle\bigcap_{1\leq p\leq n} \mathcal{A}_{i_p,j_{p+1},p}| &= 2 + (\textstyle\sum_{1\leq p\leq n} d(i_p, i_{p+1})) + 4n(1+2n) \\ &\leq 2 + k + 4n(1+2n) \end{aligned}$$

So these first n intersections straightforwardly encode the tour in the TSP instance. Now, observe that $A \cap T = \emptyset$ because every C_i from its first component is different from C_0. Consequently if the instance of the TSP has a solution, the sequence $\mathcal{A}_{i_1,i_2,1}, \ldots, \mathcal{A}_{i_n,i_1,n}, T, S$ where S is a sequence over $\{(\mathcal{A}_{i,j,p})_{i,j,p}\} \backslash \{\mathcal{A}_{i_k,i_{k+1},k} : k \leq n\}$ is a solution to IO2.

Completeness Consider an intersection of the form

$$A = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i}) \cap T \cap (\bigcap_{(\alpha_j)_j \in [1..n]^3 \backslash I} \mathcal{A}_{\alpha_j})$$

where no intermediate automaton has a size greater than B. In particular this is true for $A' = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i})$. We note $m = |I|$ and we can deduce that:

- α_1 is of the form $(x_1, y_1, 1)$; if $\alpha_i = (x_i, y_i, p)$ then $\alpha_{i+1} = (x_{i+1}, y_{i+1}, p+1)$ and $m \leq n$, otherwise component 3 would make $|A'| > B$. This implies that α_i is of the form (x_i, y_i, i)
- if $\alpha_i = (x_i, y_i, i)$ and $\alpha_{i+1} = (x_{i+1}, y_{i+1}, i+1)$ then $y_i = x_{i+1}$ and $m = n$ implies that α_m is of the form (x_m, x_1, m). Otherwise component 2 would make $|A'| > B$
- Finally, $m \geq n$ otherwise $|A' \cap T| > B$. This implies that $m = n$. Remark that this also implies that $|A' \cap T| = 0$.

In other words A' encodes a tour $i_1, i_2, \ldots, i_n, i_1$ in our instance of the TSP. Furthermore the size of A' if we follow its construction as stated above is

$$\begin{aligned} |A'| = |\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| &\leq B \\ |\mathcal{C}_1| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| &\leq B \\ |\mathcal{C}_1| + 4n + 2 + 2n(4n-1) &\leq B \\ |\mathcal{C}_1| + 2 + 2n(1+4n) &\leq 2 + k + 4n(1+2n) \\ |\mathcal{C}_1| &\leq k + 2n \\ 2n + d(i_1, i_2) + \cdots + d(i_n, i_1) &\leq k + 2n \\ d(i_1, i_2) + \cdots + d(i_n, i_1) &\leq k \end{aligned}$$

And so the tour is actually a solution for our instance of the TSP.

Theorem 3. ***(IO3)*** *is NP-complete.*

Proof. The encoding remains the same that the one for (IO2) except for the first component. The non-instantiated distances before position p are erased by intersection with $C_\perp$. (Notice that $C\perp \cap C_{i\in\{0,1,2\}} = C_\perp$)

$$\mathcal{C}_{1,i,j,p} = (V(C_\perp + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{p-1}VC_{d(i,j)}(V(C_0 + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{n-p}V(\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p})$$

The bound for this third problem is $B = k + n(2 + 2n + |\mathcal{C}_2| + |\mathcal{C}_3|) = k + n(2 + 8n + 8n^2)$ which corresponds to k and the part of the first automaton of the tour that does not disappear by intersection before the intersection with T.

References

1. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. **125**(2) (1994) 315–328
2. Yu, S.: On the state complexity of combined operations. In Ibarra, O.H., Yen, H.C., eds.: CIAA. Volume 4094 of Lecture Notes in Computer Science., Springer (2006) 11–22
3. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)
4. Karakostas, G., Lipton, R.J., Viglas, A.: On the complexity of intersecting finite state automata. In: IEEE Conference on Computational Complexity. (2000) 229–234
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press (1990)
6. Moortgart, M.: Categorial Type Logics. In van Benthem, J., ter Meulen, A., eds.: Handbook of Logic and Language. Elsevier (1996)
7. Perrier, G.: La sémantique dans les grammaires d'interaction. Traitement Automatique des Langues **45**(3) (2004) 123–144
8. Kahane, S.: Polarized unification grammars. In: ACL, The Association for Computer Linguistics (2006)
9. Kupiec, J.: Robust part-of-speech tagging using a hidden Markov model. Computer Speech and Language **6** (1992) 225–242
10. Merialdo, B.: Tagging English text with a probabilistic model. Computational Linguistics **20**(2) (1994) 155–172
11. Weischedel, R., Meteer, M., Schwarz, R., Ramshaw, L.A., Palmucci, J.: Coping with ambiguity and unknown words through probabilistic models. Computational Linguistics **19**(2) (1993) 155–172
12. Bonfante, G., Guillaume, B., Perrier, G.: Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. (2004)
13. Tapanainen, P.: Applying a Finite-State Intersection Grammar. In: Finite-State Language Processing. MIT (1997)

A Appendix

A.1 A disambiguation example

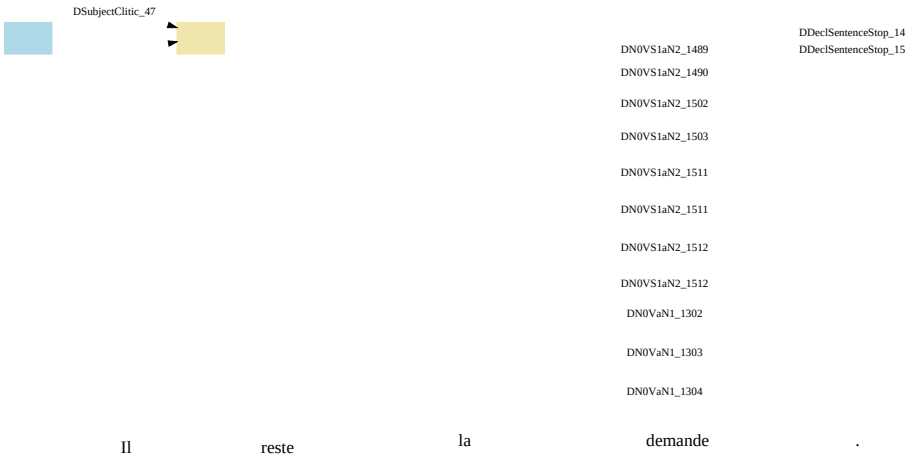


Fig. 1. Initial automaton provided by an IG for the sentence *"Il reste la demande."*. There are initially 7392 way of tagging the sentence.

The following table illustrates the fact that the choice of the order has some influence on the performance of the computation of the intersection. The first two column correspond to the choice of the feature name and the feature value. The third/fourth/fifth colums give the number of states (number of paths and number of edges) for this choice. The sixth column show the number of states of the intersection of automata above the current line and the last colum gives the number of states of the intersection of automata below the current line.

A.2 Some automata for the reduction

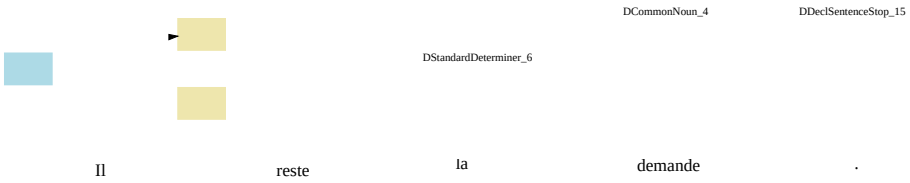


Fig. 2. Final automaton after every filtering step has been performed. Only XXX tagging candidates remain.

f	v	$states$	$paths$	$edges$	$intersection \downarrow$	$intersection \uparrow$
		6	7392	45		
$break$	$close$	6	3696	44	6	8
cat	pp	6	2688	31	6	8
cat	ap	6	5808	42	6	8
cpl	que	6	5760	41	6	8
cpl	$ou\|pourquoi\|quand\|si$	6	5376	39	6	8
cpl	de	6	6048	41	6	12
$prep$	a	6	2688	31	6	12
$reflex$	$true$	6	6720	43	6	12
cat	s	8	3984	49	8	12
cat	$n\|np$	13	2334	70	10	18
$funct$	$subj\|obj\|attr$	12	800	61	8	12
$total$		87			74	118

Table 1. Sizes of filtering automata for *"Il reste la demande."*

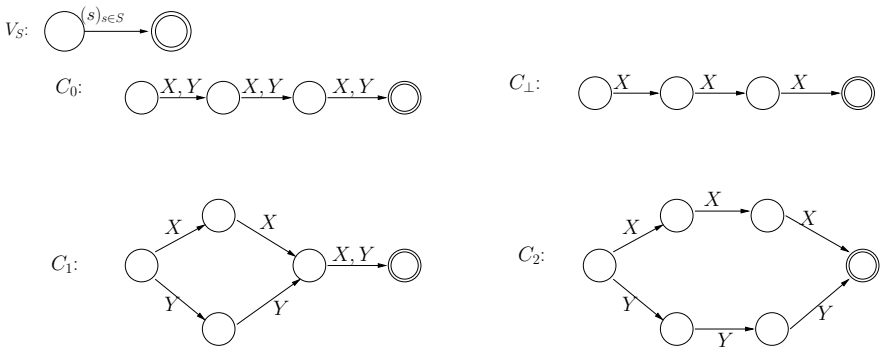


Fig. 3. some brick automata

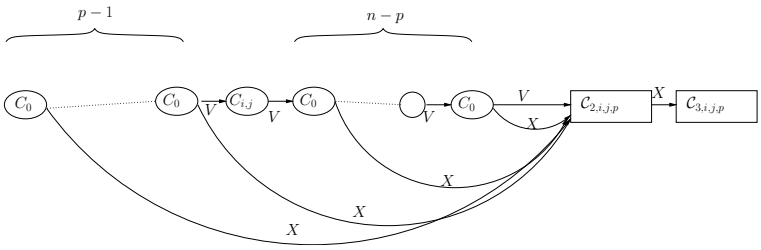
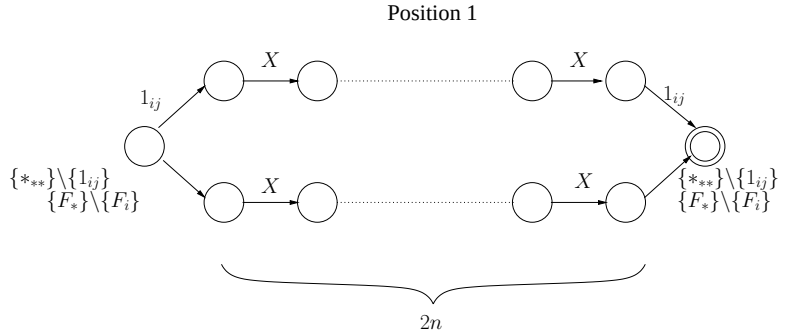
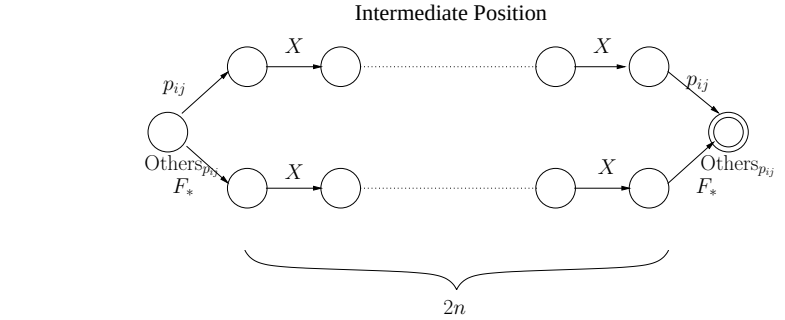


Fig. 4. automaton $\mathcal{A}_{i,j,p}$ with detailed first component





$$\text{Others}_{p_{ij}} = \{*_{**}\}\backslash\{p_{ij}\}\backslash\{p'_{*i} | p' = p-1\}$$

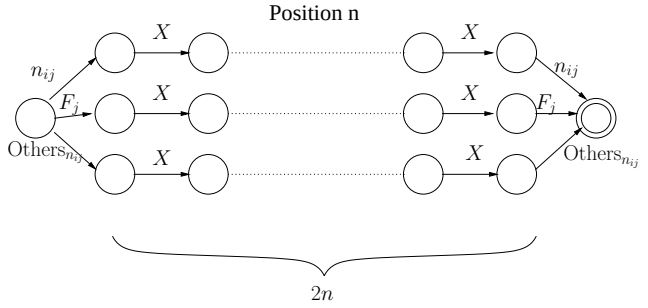


Fig. 5. the second component for the automaton $\mathcal{A}_{i,j,p}$

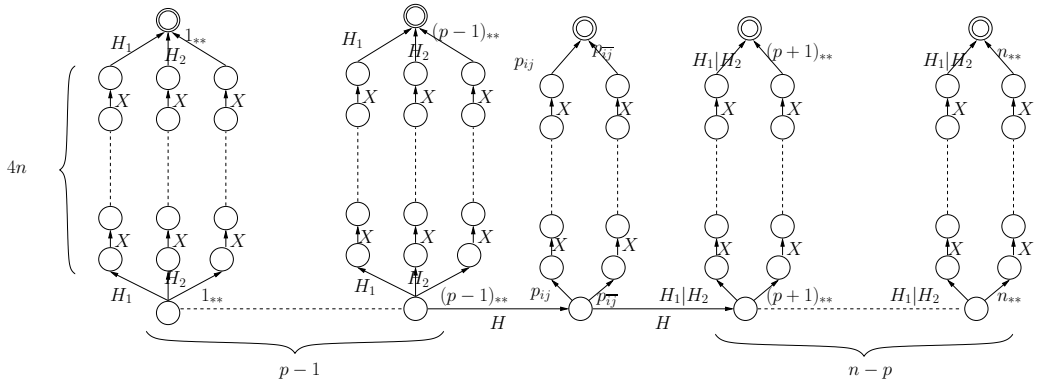


Fig. 6. the third component for the automaton $\mathcal{A}_{i,j,p}$

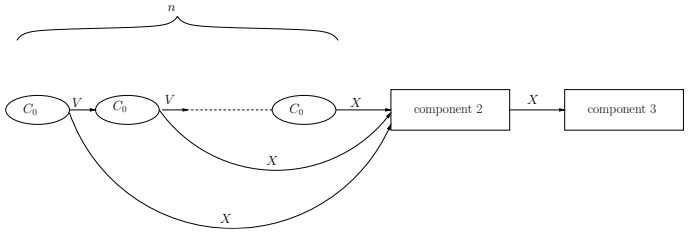


Fig. 7. the automaton T